Day 1: Report Repair

(Povezava na nalogo)

Vhodna datoteka vsebuje števila; po eno v vrstici. Prvi del naloge je poiskati dve števili, katerih vsota je 2020, in izpisati njun produkt. Drugi del je poiskati tako trojko in poiskati njen produkt.

Števila bomo prebrali v množico; ta nas bo kasneje pripeljala do najhitrejše rešitve.

```
numbers = {int(x) for x in open("input.txt")}
```

Naivna rešitev: dve zanki

Naivna rešitev je, da pač preverimo vse pare.

```
for x in numbers:
    for y in numbers:
        if x + y == 2020:
            print(x * y)
```

```
270144
270144
```

Čeprav obstaja le en tak par, se izpiše dvakrat, to pa zato, ker dejansko vsak par preskusimo dvakrat, z `x` in `y` v zamenjanih vlogah. Če bi za `print` dodali `break`, ne bi dosegli ničesar, ker bi prekinil le notranjo zanko, ne pa tudi zunanje. V Pythonu se to ne da. Poleg tega pa bi še vedno pregledovali vsak par dvakrat. Običajna pot, da se temu izognemo, je, da spustimo notranjo zanko samo do tam, do koder je prišla prva. Študenti, ki znajo programirati že od prej, tu radi podležejo skušnjavi, da bi uporabili `range(len(numbers))`. A ni potrebno, saj imamo `enumerate`. V principu bi torej naredili

```
for i, x in enumerate(numbers):
    for y in numbers[:i]:
        if x + y == 2020:
            print(x * y)
```

vendar ne bomo, ker

- `numbers` ni seznam, temveč množica;
- bi, če bi bil seznam, na ta način stalno delali kopijo tega seznama in tako nismo ničesar pridobili.

To se da obvoziti, vendar ni preveč zanimivo. Nadaljevali bomo kar z neučinkovito rešitvijo; spodaj bomo itak naredili hitreje.

Tule nas zanima le še, ali je to možno narediti z generatorjem. Stvar je preprosta: zanimajo nas vsi produkti števil, katerih vsota je enaka 2020. Iskani generator je torej

```
(x * y for x in numbers for y in numbers if x + y == 2020)
```

```
<generator object <genexpr> at 0x7fd9638524d0>
```

Prva poučna stvar je, da generator prek parov naredimo tako, da vanj vpišemo dve zanki.

Druga poučna stvar bo, kako dobiti prvi element generatorja: z `next` pač.

```
next(x * y for x in numbers for y in numbers if x + y == 2020)
```

```
270144
```

```
next(x * y * z for x in numbers for y in numbers for z in numbers if x + y + z == 2020)
```

```
261342720
```

Če imamo n števil, bo rešitev prvega dela pregledala n^2 parov, rešitev drugega pa n^3 parov. Če bi bilo števil desetkrat več, bi za prvi del potrebovali stokrat, za drugega pa tisočkrat več časa. Temu pravimo, da ima program kvadratno oz. kubično časovno zahtevnost.

Gre boljše?

Rešitev z množicami

Ko imamo `x` pravzaprav vemo, kakšen je pripadajoči `y`: `2020 - x`, pač. Namesto da gremo z notranjo zanko ponovno prek množice, lahko preprosto preverimo, ali množica vsebuje `2020 - x`.

V "razpisani" varianti je to videti tako:

```
for x in numbers:
    if 2020 - x in numbers:
        print(x * (2020 - x))
        break
```

```
270144
```

Z generatorjem pa tako:

```
next(x * (2020 - x) for x in numbers if 2020 - x in numbers)
```

```
270144
```

Zanka je le ena, preverimo le `x` števil, čas preverjanja pogoja `2020 - x in numbers` pa je neodvisen od števila števil, saj je `numbers` množica. (Če bi imeli namesto množice seznam, pa bi bil čas primerjanja sorazmeren njegovi velikosti, torej ne bi pridobili ničesar!) Čas izvajanja je torej sorazmeren številu števil; takemu času pravimo, da je linearen.

Za drugi del naloge potrebujemo dve zanki - namesto treh, ki smo jih imeli prej.

```
for x in numbers:
    for y in numbers:
```

```
        if 2020 - x - y in numbers:
            print(x * y * (2020 - x - y))
            break
```

```
261342720
261342720
261342720
```

Rezultat se izpiše trikrat, ker x in y prevzameta po dve števili iz trojke, kar lahko naredita na tri načine. Z generatorjem bomo pobrali le prvo rešitev.

```
next(x * y * (2020 - x - y) for x in numbers for y in numbers if 2020 - x - y in numbers)
```

```
261342720
```

Gre hitreje?

Ni dokazano, da ne gre. Odprt problem.

Kot je napisal asistent pri predmetu, Tomaž Hočevar,

> Za drugi del ni bistveno hitrejšega načina od te in podobnih rešitev s kvadratno časovno zahtevnostjo. Problem je znan pod imenom 3SUM. Ali obstaja rešitev s časovno zahtevnostjo $O(n^x)$, kjer je $x < 2$, je še nerešen problem. Je pa možno (v teoriji) malenkost izboljšati kvadratno rešitev (če koga zanima, ima članek naslov "Threesomes, Degenerates, and Love Triangles").